The State Operator in Real Time Process Algebra

J.C.M. Baeten
Department of Software Technology, CWI,
P.O.Box 4079, 1009 AB Amsterdam, The Netherlands
and
Programming Research Group, University of Amsterdam,
P.O.Box 41882, 1009 DB Amsterdam, The Netherlands

J.A. Bergstra
Programming Research Group, University of Amsterdam,
P.O.Box 41882, 1009 DB Amsterdam, The Netherlands
and
Department of Philosophy, Utrecht University,
Heidelberglaan 2, 3584 CS Utrecht, The Netherlands

Abstract: We extend the real time process algebra of [BB91a] with the state operator of [BB88]. We show the usefulness of this extension in several examples. We use concepts from (classical) real space process algebra of [BB91b] in order to deal with different locations.

Key words & Phrases: process algebra, real time, state operator, locations.
Note: Partial support received by ESPRIT basic research action 3006, CONCUR, and by RACE contract 1046, SPECS. This document does not necessarily reflect the views of the SPECS consortium.

Contents:

1. INTRODUCTION

We provide an extension of real time process algebra of [BB91a] with a state operator. For this purpose it is plausible to use a version of real time process algebra with located actions. Departing from a finite set A of action names and a finite set L of location names, a version of the real space process algebra of [BB91b] is developed. As in [BB91a], timed deadlocks are used instead of the untimed deadlock of [BB91b]. Further, an operational semantics is given in the style of KLUSENER [K91], which is a modification of the operational semantics of [BB91a].

The work on extensions of process algebra that encorporate notions of time started with the work of REED & ROSCOE (e.g. [RR88]), who discuss an extension of CSP (of [H85]). In 1989, we presented our extension of ACP (see [BB91a]). Extensions of CCS (of [M80, M89]) followed, see e.g. MOLLER & TOFTS [MT90], JEFFREY [J91a]. The real time process algebra as presented in [BB91a] uses a dense time domain, affixes timestamps to all atomic actions, and uses an integral operator to express that an action occurs within a certain time interval.

In [BB91b], we extended our previous work to take space coordinates into account. This leads to a relativistic calculus where events are not totally ordered by time any longer. This idea is also taken up in JEFFREY [J91b], MURPHY [MU91]. In the present paper, we limit the use of space to a finite set of *locations*.

This extension of real time process algebra is not as straightforward as one might expect because the effect of actions with components at various locations (multi-actions) on a state has to be evaluated in some predefined order. This reflects the fact that it is difficult to have a notion of a global state in a distributed system. In order to solve the problem, we assume an ordering on the locations and require that multi-actions are evaluated in increasing order (if possible). One might expect it to be more natural to require that actions can be effectuated in arbitrary order, resulting in the same state, but that will exclude many useful applications of the state operator.

Nevertheless we consider our combination of a state operator and classical real space process algebra to be satisfactory. We notice that adding the state operator to the relativistic real space process algebra of [BB91b] is difficult if not impossible.

Finally, we remark that we only consider *concrete* process algebra here: there is no concept of a silent or empty step.

2. REAL TIME PROCESS ALGEBRA

We start with a review of real time process algebra as introduced in [BB91a], but instead of the operational semantics given there, we use the variant of KLUSENER [K91].

2.1 BASIC PROCESS ALGEBRA

Process algebra (see [BK84, BK86, BW90]) starts from a given *action alphabet* A (usually finite). Elements a,b,c of A are called *atomic actions*, and are constants of the sort P of *processes*. The theory Basic Process Algebra (BPA) has two binary operators +,·: $P \times P \to P$; + stands for alternative composition and · for sequential composition. BPA has the axioms from table 1.

$X + Y = Y + X$	A1
$(X + Y) + Z = X + (Y + Z)$	A2
$X + X = X$	A3
$(X + Y) \cdot Z = X \cdot Z + Y \cdot Z$	A4
$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	A5

Table 1. BPA.

If we add to BPA a special constant δ in P (not in A) standing for *inaction*, comparable to NIL or 0 of CCS (see MILNER [M80, 89] or HENNESSY [HE88]) or STOP of CSP (see HOARE [H85]), we obtain the theory BPAδ. The two axioms for δ are in table 2.

$X + \delta = X$	A6
$\delta \cdot X = \delta$	A7

Table 2. BPAδ = BPA + A6, A7.

When we add real time to this setting, our basic actions are not from the set $A_\delta = A \cup \{\delta\}$, but from the set

$$AT = \{a(t) \mid a \in A_\delta, t \in \mathbb{R}^{\geq 0}\} \cup \{\delta\}.$$

Here, $\mathbb{R}^{\geq 0} = \{r \in \mathbb{R} \mid r \geq 0\}$. The process $a(t)$ performs action a at time t, and then terminates. The process $\delta(t)$ deadlocks at time t. The process δ cannot do anything, in particular it cannot wait. Again, these actions can be combined by $+,\cdot$. We have the identification $\delta(0) = \delta$.

The letter A in the names of the following axioms refers to *absolute time* (versions with relative time were also considered in [BB91a], but are not treated here).

As in [BB91a], we have the additional operation $\gg$, the *(absolute) time shift*. $t \gg X$ denotes the process X starting at time t. This means that all actions that have to be performed at or before time t are turned into deadlocks because their execution has been delayed too long.

In table 3, we have $a \in A_\delta$.

$a(0) = \delta(0)$	ATA1
$\delta(t) \cdot X = \delta(t)$	ATA2
$t < r \Rightarrow \delta(t) + \delta(r) = \delta(r)$	ATA3
$a(t) + \delta(t) = a(t)$	ATA4
$a(t) \cdot X = a(t) \cdot (t \gg X)$	ATA5
$t < r \Rightarrow t \gg a(r) = a(r)$	ATB1
$t \geq r \Rightarrow t \gg a(r) = \delta(t)$	ATB2
$t \gg (X + Y) = (t \gg X) + (t \gg Y)$	ATB3
$t \gg (X \cdot Y) = (t \gg X) \cdot Y$	ATB4

Table 3. BPAρδ.

A *closed process expression* (CPE) over the signature of BPAρδ with atoms A is an expression that does not contain variables for atoms, processes or real numbers. We allow every real number as a constant, which means there are uncountably many such closed process expressions. For finite closed process expressions an initial algebra can be defined. This is the initial algebra model of BPAρδ. This structure identifies two closed expressions whenever these can be shown identical by means of application of the axioms. This definition of closed process expressions and an initial model can be extended to all extensions of BPA that are described below.

We will look at an operational model next. We denote the set of actions over A without variables by IA (the set of instantiated actions), and write IAδ when we use the set A_δ.

2.2 OPERATIONAL SEMANTICS.

We describe an operational semantics for BPAρδ following KLUSENER [K91]. His operational semantics is a simplification of the one in [BB91a]. In fact the operational semantics of [K91] is more abstract than the one given in [BB91a]. We have two relations

$$\text{step} \subseteq \text{CPE} \times \text{IA} \times \text{CPE}$$
$$\text{terminate} \subseteq \text{CPE} \times \text{IA}\delta.$$

The extension of the relations for atomic actions (so excluding δ-termination) is found as the least fixed point of a simultaneous inductive definition. We write

$x \xrightarrow{a(r)} x'$	for	step(x, a(r), x'), and
$x \xrightarrow{a(r)} \surd$	for	terminate(x, a(r)).

Notice that in the first case $a \in A$ and in the second case $a \in A_\delta$.

The inductive rules for the operational semantics are similar to those used in structural operational semantics. We list the rules for atomic actions. In table 4, we have $a \in IA$, $r,s > 0$ (we never allow timestamp 0!), $x,x',y \in CPE$.

$$r > 0 \Rightarrow a(r) \xrightarrow{a(r)} \surd$$

$$\frac{x \xrightarrow{a(r)} x'}{x+y \xrightarrow{a(r)} x',\ y+x \xrightarrow{a(r)} x'}$$

$$\frac{x \xrightarrow{a(r)} \surd}{x+y \xrightarrow{a(r)} \surd,\ y+x \xrightarrow{a(r)} \surd}$$

$$\frac{x \xrightarrow{a(r)} x'}{x \cdot y \xrightarrow{a(r)} x' \cdot y} \qquad \frac{x \xrightarrow{a(r)} \surd}{x \cdot y \xrightarrow{a(r)} r \gg y}$$

$$\frac{x \xrightarrow{a(r)} x',\ r > s}{s \gg x \xrightarrow{a(r)} x'} \qquad \frac{x \xrightarrow{a(r)} \surd,\ r > s}{s \gg x \xrightarrow{a(r)} \surd}$$

Table 4. Action rules for atomic actions for BPAρδ.

2.3 Ultimate Delay

In order to state the rule for deadlock actions, we need an auxiliary notion: $U(x)$ is the *ultimate delay* of x. This notion is defined as follows (table 5, $a \in A$).

The ultimate delay operator U takes a process expression X in CPE, and returns an element of $\mathbb{R}^{\geq 0}$. The intended meaning is that X can idle before $U(X)$, but X can never reach time $U(X)$ or a later time by just idling.

$U(a(t)) = t$	ATU1
$U(\delta(t)) = t$	ATU2
$U(X + Y) = \max\{U(X), U(Y)\}$	ATU3
$U(X \cdot Y) = U(X)$	ATU4
$U(t \gg X) = \max\{t, U(X)\}$	ATU5

Table 5. Ultimate delay operator.

Now we construct a transition system for a term as follows: first generate all transitions involving atomic actions using the inductive rules of table 4. Then, for every node (term) p in this transition system we do the following: first determine its ultimate delay $U(p) = u$ by means of table 5. Then, if the ultimate delay is larger than the supremum of the time stamps of all outgoing transitions, we add a transition

$$p \xrightarrow{\delta(u)} \surd.$$

Otherwise, we do nothing (add no transitions).

2.4 Bisimulations

Again we consider the class CPE of closed process expressions over BPAρδ. A *bisimulation* on CPE is a binary relation R such that

i. for each p and q with R(p, q): if there is a step a(s) possible from p to p', then there is a CPE q' such that R(p', q') and there is a step a(s) possible from q to q'.

ii. for each p and q with R(p, q): if there is a step a(s) possible from q to q', then there is a CPE p' such that R(p', q') and there is a step a(s) possible from p to p'.

iii. for each p and q with R(p, q): a termination step a(s) or δ(s) is possible from p iff it is possible from q.

We say expressions p and q are *bisimilar*, denoted $p \leftrightarrow q$, if there exists a bisimulation on CPE with R(p,q). In [K91] it is shown that bisimulation is a congruence relation on CPE, and that CPE/$\leftrightarrow$ is a model for BPAρδ. Indeed, this model is isomorphic to the initial algebra. The advantage of this operational semantics is, that it allows extensions to models containing recursively defined processes.

Next, we extend the system BPAρδ with an operator for parallel composition.

2.5 Operational semantics for parallel composition.

$$\frac{x \xrightarrow{a(r)} x',\ r < U(y)}{x \| y \xrightarrow{a(r)} x' \| (r \gg y),\ y \| x \xrightarrow{a(r)} (r \gg y) \| x'}$$

$$\frac{x \xrightarrow{a(r)} \surd,\ r < U(y)}{x \| y \xrightarrow{a(r)} r \gg y,\ y \| x \xrightarrow{a(r)} r \gg y}$$

$$\frac{x \xrightarrow{a(r)} x',\ y \xrightarrow{b(r)} y',\ a \mid b = c \neq \delta}{x \| y \xrightarrow{c(r)} x' \| y'}$$

$$\frac{x \xrightarrow{a(r)} x',\ y \xrightarrow{b(r)} \surd,\ a \mid b = c \neq \delta}{x \| y \xrightarrow{c(r)} x',\ y \| x \xrightarrow{c(r)} x'}$$

$$\frac{x \xrightarrow{a(r)} \surd,\ y \xrightarrow{b(r)} \surd,\ a \mid b = c \neq \delta}{x \| y \xrightarrow{c(r)} \surd}$$

Table 6. Action rules for parallel composition.

Now we extend the system BPA$\rho\delta$ with the parallel composition operator $\|$ (merge). We assume we have given a communication function $| : A_\delta \times A_\delta \rightarrow A_\delta$. $|$ is commutative, associative and δ is a zero element for it. The operational semantics for atomic actions is given in table 6. With the additional rules of table 6, a transition system is generated. We then add δ-rules as before.

We see that the action rules for parallel composition make use of the ultimate delay operator. This operator was introduced in 2.3. We add rules so that the ultimate delay can be syntactically determined for every term.

$U(X \| Y) = \min\{U(X), U(Y)\}$	ATU6
$U(X \mathbin{\|\!\lfloor} Y) = \min\{U(X), U(Y)\}$	ATU7
$U(X \mid Y) = \min\{U(X), U(Y)\}$	ATU8
$U(X \gg t) = \min\{t, U(X)\}$	ATU9
$U(\partial_H(X)) = U(X)$	ATU10

Table 7. Ultimate delay with parallel composition.

In table 7, we also find a number of auxiliary operators that are needed to give a axiomatic characterization of parallel composition (cf. [BB91a]).

2.6 Bounded Initialization

The bounded initialization operator is also denoted by $\gg$, and is the counterpart of the operator with the same name that we saw in the axiomatization of BPA$\rho\delta$. With $X \gg t$ we denote the process X with its behaviour restricted to the extent that its first action must be performed at a time before $t \in \mathbb{R}^{\geq 0}$. Axioms defining $\gg$ are in table 8, where we have $a \in A_\delta$.

$r \geq t \Rightarrow a(r) \gg t = \delta(t)$	ATB5
$r < t \Rightarrow a(r) \gg t = a(r)$	ATB6
$(X + Y) \gg t = (X \gg t) + (Y \gg t)$	ATB7
$(X \cdot Y) \gg t = (X \gg t) \cdot Y$	ATB8

Table 7. Bounded initialization operator.

2.7 Algebra of Communicating Processes.

Apart from the ultimate delay and bounded initialization operators, an axiomatization of parallel composition also uses the left merge operator $\mathbin{\|\!\lfloor}$ and communication merge operator $|$ of [BK84]. Let H be some subset of A, and let a,b,c be elements of A_δ.

$a \mid b = b \mid a$	C1
$a \mid (b \mid c) = (a \mid b) \mid c$	C2
$\delta \mid a = \delta$	C3
$t \neq r \;\Rightarrow\; a(t) \mid b(r) = \delta(\min(t,r))$	ATC1
$a(t) \mid b(t) = (a \mid b)(t)$	ATC2
$X \parallel Y = X \mathbin{\lfloor\!\lfloor} Y + Y \mathbin{\lfloor\!\lfloor} X + X \mid Y$	CM1
$a(t) \mathbin{\lfloor\!\lfloor} X = (a(t) \gg U(X)) \cdot X$	ATCM2
$(a(t) \cdot X) \mathbin{\lfloor\!\lfloor} Y = (a(t) \gg U(Y)) \cdot (X \parallel Y)$	ATCM3
$(X + Y) \mathbin{\lfloor\!\lfloor} Z = X \mathbin{\lfloor\!\lfloor} Z + Y \mathbin{\lfloor\!\lfloor} Z$	CM4
$(a(t) \cdot X) \mid b(r) = (a(t) \mid b(r)) \cdot X$	CM5'
$a(t) \mid (b(r) \cdot X) = (a(t) \mid b(r)) \cdot X$	CM6'
$(a(t) \cdot X) \mid (b(r) \cdot Y) = (a(t) \mid b(r)) \cdot (X \parallel Y)$	CM7'
$(X + Y) \mid Z = X \mid Z + Y \mid Z$	CM8
$X \mid (Y + Z) = X \mid Y + X \mid Z$	CM9
$\partial_H(a) = a \qquad \text{if } a \notin H$	D1
$\partial_H(a) = \delta \qquad \text{if } a \in H$	D2
$\partial_H(a(t)) = (\partial_H(a))(t)$	ATD
$\partial_H(X + Y) = \partial_H(X) + \partial_H(Y)$	D3
$\partial_H(X \cdot Y) = \partial_H(X) \cdot \partial_H(Y)$	D4

Table 8. ACPρ = BPA$\rho\delta$ + ATU1-4 + ATB5-8 + C1-3 +
+ ATC1,2 + CM1,4,8,9 + CM5',6',7' + ATCM2,3 + D1-4 + ATD.

2.8 Operational semantics

We can also give action rules for the auxiliary operators. The rules are straightforward.

$\dfrac{x \xrightarrow{a(r)} x',\ r < t}{x \gg t \xrightarrow{a(r)} x'}$	$\dfrac{x \xrightarrow{a(r)} \surd,\ r < t}{x \gg t \xrightarrow{a(r)} \surd}$
$\dfrac{x \xrightarrow{a(r)} x',\ r < U(y)}{x \lfloor\!\lfloor y \xrightarrow{a(r)} x' \parallel (r \gg y)}$	$\dfrac{x \xrightarrow{a(r)} \surd,\ r < U(y)}{x \lfloor\!\lfloor y \xrightarrow{a(r)} r \gg y}$
$\dfrac{x \xrightarrow{a(r)} x',\ y \xrightarrow{b(r)} y',\ a \mid b = c \neq \delta}{x \mid y \xrightarrow{c(r)} x' \parallel y'}$	
$\dfrac{x \xrightarrow{a(r)} x',\ y \xrightarrow{b(r)} \surd,\ a \mid b = c \neq \delta}{x \mid y \xrightarrow{c(r)} x',\ y \mid x \xrightarrow{c(r)} x'}$	
$\dfrac{x \xrightarrow{a(r)} \surd,\ y \xrightarrow{b(r)} \surd,\ a \mid b = c \neq \delta}{x \mid y \xrightarrow{c(r)} \surd}$	
$\dfrac{x \xrightarrow{a(r)} x',\ a \notin H}{\partial_H(x) \xrightarrow{a(r)} \partial_H(x')}$	$\dfrac{x \xrightarrow{a(r)} \surd,\ a \notin H}{\partial_H(x) \xrightarrow{a(r)} \surd}$

Table 9. Action relations for auxiliary operators of ACPρ.

2.9 INTEGRATION

An extension of ACPρ (called ACPρI) that is very useful in applications is the extension with the integral operator, denoting a choice over a continuum of alternatives. I.e., if V is a subset of $\mathbb{R}^{\geq 0}$, and v is a variable over $\mathbb{R}^{\geq 0}$, then $\int_{v \in V} P$ denotes the alternative composition of alternatives $P[t/v]$ for $t \in V$ (expression P with nonnegative real t substituted for variable v). For more information, we refer the reader to [BB91a] and [K91]. The operational semantics is straightforward (table 10).

$\dfrac{x(t) \xrightarrow{a(r)} x',\ t \in V}{\int_{v \in V} x(v) \xrightarrow{a(r)} x'}$	$\dfrac{x(t) \xrightarrow{a(r)} \surd,\ t \in V}{\int_{v \in V} x(v) \xrightarrow{a(r)} \surd}$

Table 10. Action relations for integration.

We will not provide axioms for the integral operator here (and refer the reader to [BB91a] and [K91]), except for the axiom for the ultimate delay operator:

$$U\left(\int_{v \in V} P\right) = \sup\{U(P[t/v]) : t \in V\} \qquad \text{ATU11.}$$

2. 10 GRAPH MODEL

It is possible to construct a graph model for ACPρI. However, we obtain a number of simplifications if we only consider the domain of process *trees*. Therefore, we will limit our domain to trees.

Process trees are directed rooted trees with edges labeled by timed atomic or delta actions, satisfying the condition that for each pair of consecutive transitions $s_1 \xrightarrow{a(r)} s_2 \xrightarrow{b(t)} s_3$ it is required that $r < t$ (however, in case $b \equiv \delta$, we also allow $r = t$). Moreover, we require that the endnode of a δ-transition has no outgoing edges, and that the timestamp of a δ-transition is larger than the supremum of the timestamps of its brother edges (edges starting from the same node).

Now $+, \cdot, \|, \lfloor\!\lfloor, \mid, \partial_H, \gg, U, \gg$ and $\int$ can be defined on these trees in a straightforward manner:

- For +, take the disjoint union of the trees and identify the roots. If one of the roots has an outgoing δ-edge, remove this edge if its timestamp is less than or equal than the supremum of the timestamps of the outgoing edges of the other root (but keep one of the two δ-edges, if both roots have a δ-edge with the same timestamp).
- $t \gg g$ is obtained by removing every edge from the root with label $a(r)$ with $r \leq t$. If this removes all edges starting from the root, add a $\delta(t)$-edge to an endpoint.
- $g \cdot h$ is constructed as follows: identify each non-δ endpoint s of g (endpoint with no incoming δ-edge) with the root of a copy of $t \gg h$, where t is the time of the edge leading to s.
- $U(g) = \sup\{r \in \mathbb{R}^{\geq 0} \mid root(g) \xrightarrow{a(r)} s$ for $a \in A_\delta$ and certain $s \in g\}$. If g is the trivial one-node graph, put $U(g) = \infty$.
- Let for $s \in g$, $(g)_s$ denote the subgraph of g with root s. Then $g \| h$ is defined as follows:
- the set of states is the cartesian product of the state sets of g and h, the root the pair of roots.
- transitions: if $s \xrightarrow{a(r)} s'$, $a \neq \delta$ and $r < U((h)_t)$ then $\langle s,t \rangle \xrightarrow{a(r)} \langle s',t \rangle$;
  if $t \xrightarrow{a(r)} t'$, $a \neq \delta$ and $r < U((g)_s)$ then $\langle s,t \rangle \xrightarrow{a(r)} \langle s,t' \rangle$;
  if $s \xrightarrow{a(r)} s'$ and $t \xrightarrow{b(r)} t'$ and $a \mid b = c \neq \delta$ then $\langle s,t \rangle \xrightarrow{c(r)} \langle s',t' \rangle$.

  Lastly, if a node $\langle s,t \rangle$ with s and t not both endnodes in g resp. h does not have an outgoing edge with timestamp equal to $t = \min\{U((g)_s, (h)_t\}$, we add a transition $\langle s,t \rangle \xrightarrow{\delta(t)}$ to an endpoint.

It is an exercise to show that this construction always yields a tree again.

- the construction of $g \lfloor\!\lfloor h$, $g \mid h$ and $\partial_H(g)$ is now straightforward.
- Finally, the graph of $\int_{v \in V} P$ is constructed by first identifying the roots of the graphs $P[t/v]$ for $t \in V$. Next, remove all δ-edges that do not satisfy the condition above (i.e. its timestamp is not larger than the supremum of the timestamps of its brother edges). Add again a δ-edge with timestamp t, if the ultimate delay t of the first graph is larger than the supremum of the timestamps of the remaining edges.

As an example, notice that the graph of $\int_{v \in (0,1)} \delta(v)$ only has one edge, with label $\delta(1)$.

Bisimulation on these graphs is defined as e.g. in BERGSTRA & KLOP [BK84]. One may prove in a standard fashion that bisimulation is a congruence for all operators of ACPρI.

3. LOCATIONS

We get a straightforward extension of the theory in section 2, if we add a location coordinate to all atomic actions except δ. We use a *finite* set of locations L (the use of an infinite set of locations was discussed in [BB91b]; here, we do not need that). The set of timed located atomic actions, ALT is now generated by

$\{a(\ell;t) \mid a \in A, \ell \in L, t \in \mathbb{R}^{\geq 0}\} \cup \{\delta(t) \mid t \geq 0\}$.

It will be useful to consider also the set of *untimed* located actions, i.e. the set AL generated by

$\{a(\ell) \mid a \in A, \ell \in L\} \cup \{\delta\}$.

We can draw the following picture of the algebraic signature (fig. 1).

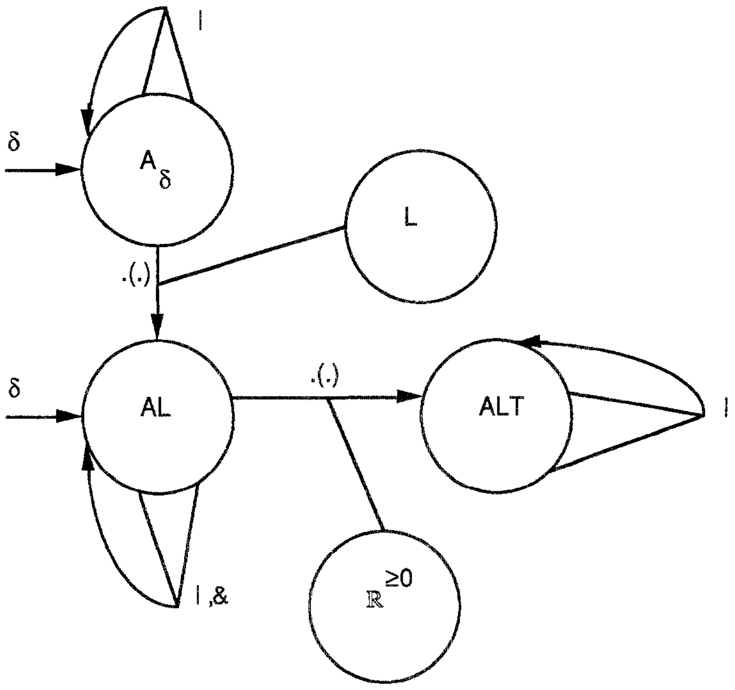

FIGURE 1.

We use $a(\ell)(t)$ as an alternative notation for $a(\ell;t)$.

3.1 MULTI-ACTIONS

Multi-actions are process terms generated by located actions and the *synchronisation function* &. Multi-actions contain actions that occur synchronously at different locations. For α,β,γ elements of AL, we have the following conditions on the synchronisation function (table 11). Further, $\ell \in L$, $a,b \in A$.

$\alpha \,\&\, \beta = \beta \,\&\, \alpha$	LO1
$\alpha \,\&\, (\beta \,\&\, \gamma) = (\alpha \,\&\, \beta) \,\&\, \gamma$	LO2
$\delta \,\&\, \alpha = \delta$	LO3
$a(\ell) \,\&\, b(\ell) = \delta$	LO4
$\delta(\ell) = \delta$	LO5

Table 11. Synchronisation function for located actions.

Using the axioms of table 11, each multi-action can be reduced to one of the following two forms:

- δ,

- $a_1(\ell_1)$ & ... & $a_n(\ell_n)$, with all locations ℓ_i different, all $a_i \in A$.

Next, we have the *communication function* $|$. As usual, and as required by axioms C1-3 in table 8, we assume that a communication function is given on atomic actions, that is commutative, associative, and has δ as a neutral element. We have the following axioms. In table 12, $a,b,c \in A$, $\alpha,\beta,\gamma \in AL$. In order to state axiom CL7, we need an auxiliary function locs, that determines the set of locations of a multi-action.

$a \mid b = b \mid a$	C1
$a \mid (b \mid c) = (a \mid b) \mid c$	C2
$\delta \mid a = \delta$	C3
$\alpha \mid \beta = \beta \mid \alpha$	CL1
$\alpha \mid (\beta \mid \gamma) = (\alpha \mid \beta) \mid \gamma$	CL2
$\delta \mid \alpha = \delta$	CL3
$a(\ell) \mid b(\ell) = (a \mid b)(\ell)$	CL4
$a(\ell) \mid (b(\ell) \,\&\, \beta) = (a \mid b)(\ell) \,\&\, \beta$	CL5
$(a(\ell) \,\&\, \alpha) \mid (b(\ell) \,\&\, \beta) = (a \mid b)(\ell) \,\&\, (\alpha \mid \beta)$	CL6
$\mathrm{locs}(\alpha) \cap \mathrm{locs}(\beta) = \varnothing \Rightarrow \alpha \mid \beta = \alpha \,\&\, \beta$	CL7
$\mathrm{locs}(\delta) = \varnothing$	LOC1
$\mathrm{locs}(a(\ell)) = \{\ell\}$	LOC2
$\ell \notin \mathrm{locs}(\alpha), \mathrm{locs}(\alpha) \neq \varnothing \Rightarrow$	
$\mathrm{locs}(a(\ell) \,\&\, \alpha) = \mathrm{locs}(\alpha) \cup \{\ell\}$	LOC3

Table 12. Communication function for located actions.

3.2 Timed multi-actions

It is now straightforward to extend the definition of the synchronisation and communication functions to timed multi-actions. In table 13, $\alpha,\beta \in AL$.

$t \neq s \Rightarrow \alpha(t) \mid \beta(s) = \delta(\min(t,s))$	CL8
$\alpha(t) \mid \beta(t) = (\alpha \mid \beta)(t)$	CL9

Table 13. Communication function on timed multi-actions.

3.3 Real time process algebra with locations

Real time process algebra with locations now has exactly the same axioms as real time process algebra, only the letters a,b now do not range over A respectively A_δ, but over multi-actions from AL as above.

The axioms for ultimate delay are again ATU1-11 (with in ATU1 a ranging over the larger set). Parallel composition is dealt with likewise, obtaining the axiom system ACPρl by adding axioms CM1,4-9, ATCM2,3, D1-4, ATD.

The operational semantics is just like in the temporal case, in section 2. Transitions are labeled with multi-actions, and these play exactly the same role as the timed actions in the case of ACPρ. Similarly we may define a graph model for ACPρl. In both cases bisimulation can be defined in the same way.

4. STATE OPERATOR

The state operator was introduced in BAETEN & BERGSTRA [BB88]. It keeps track of the global state of a system, and is used to describe actions that have a side effect on a state space. The state operator has showed itself useful in a range of applications, e.g. in the translation of programming or specification languages into process algebra (see VAANDRAGER [V90] or SPECS [S90]).

The state operator comes equiped with two functions: given a certain state and an action to be executed from that state, the function action gives the resulting action and the function effect the resulting state. Now, when we apply these functions to a located action, we have the obvious axioms

$$\mathrm{action}(a(\ell), s) = \mathrm{action}(a, s)(\ell) \qquad \mathrm{effect}(a(\ell), s) = \mathrm{effect}(a, s).$$

Things become more difficult if we go to multi-actions: in order to calculate the resulting global state, we need to apply the effect function to the component actions in a certain order. How is this order determined? We will assume that there is a partial order $<$ on locations, and that we can apply the effect function *only* on multi-actions that determine a totally ordered set of locations. We make this precise in the following definition.

4.1 DEFINITION

Let A be a set of atomic actions, let S be a set of states. Assume we have functions

$$\mathrm{action}: A_\delta \times S \to A_\delta \qquad \mathrm{effect}: A_\delta \times S \to S$$

such that $\mathrm{action}(\delta, s) = \delta$ and $\mathrm{effect}(\delta, s) = s$ for all $s \in S$ (we say: δ is inert).
Let L be a set of locations, and let $<$ be a partial order on L. If M is a set of locations, write $TO(M)$ if M is totally ordered by $<$, and write $\ell < M$ if $\ell < m$ for all $m \in M$. Then we define the extension of the functions action and effect to multi-actions as follows ($\alpha \in AL$).

$$\boxed{\begin{array}{l}
\mathrm{action}(a(\ell), s) = \mathrm{action}(a, s)(\ell) \\
\mathrm{effect}(a(\ell), s) = \mathrm{effect}(a, s) \\
\neg TO(\mathrm{locs}(\alpha)) \Rightarrow \mathrm{action}(\alpha, s) = \delta \\
\neg TO(\mathrm{locs}(\alpha)) \Rightarrow \mathrm{effect}(\alpha, s) = s \\
TO(\mathrm{locs}(\alpha)) \,\&\, \ell < \mathrm{locs}(\alpha) \Rightarrow \\
\quad \mathrm{action}(a(\ell) \,\&\, \alpha, s) = \mathrm{action}(a, \mathrm{effect}(\alpha, s))(\ell) \,\&\, \mathrm{action}(\alpha, s) \\
TO(\mathrm{locs}(\alpha)) \,\&\, \ell < \mathrm{locs}(\alpha) \Rightarrow \\
\qquad \mathrm{effect}(a(\ell) \,\&\, \alpha, s) = \mathrm{effect}(a, \mathrm{effect}(\alpha, s))
\end{array}}$$

Table 14. Action and effect on multi-actions.

4.2 DEFINITION

The defining equations for the state operator are now straightforward (cf. [BB88]). If S is a set of states, then for each $s \in S$ we have an operator $\lambda_s: P \to P$. In table 15, $s \in S$, $\alpha \in AL$, $t \geq 0$, x,y processes.

$\lambda_s(\alpha(t)) = \mathrm{action}(\alpha, s)(t)$	SO1
$\lambda_s(\alpha(t) \cdot x) = \mathrm{action}(\alpha, s)(t) \cdot \lambda_{\mathrm{effect}(\alpha, s)}(x)$	SO2
$\lambda_s(x + y) = \lambda_s(x) + \lambda_s(y)$	SO3

Table 15. State operator.

In case we deal with integrals, the last equation SO3 has to be extended to the following:

$$\lambda_s(\int_{v\in V} P) = \int_{v\in V} \lambda_s(P) \qquad \text{SO4.}$$

It is equally straightforward to give action rules for the operational semantics ($\alpha \in AL$, $\alpha\neq\delta$).

$\dfrac{x \xrightarrow{\alpha(r)} x',\ action(\alpha,s)=\beta\neq\delta}{\lambda_s(x) \xrightarrow{\beta(r)} \lambda_{effect(\alpha,s)}(x')}$	$\dfrac{x \xrightarrow{\alpha(r)} \surd,\ action(\alpha,s)=\beta\neq\delta}{\lambda_s(x) \xrightarrow{\beta(r)} \surd}$

Table 16. Action relations for the state operator.

In order to deal with δ-transitions, we add the axiom

$$U(\lambda_s(x)) = U(x) \qquad \text{ATU12.}$$

Then, we determine the existence of δ-transitions as before (in 2.3).

4.3 EXAMPLES

1. A clock. Suppose we have a fixed location ℓ. Define, for each $s \in \mathbb{N}$,

$effect(tick, s) = s+1$
$action(tick, s) = tick.$

Then a clock is given by the process $\lambda_0(P(0))$, where $P(t) = tick(\ell; t+1)\cdot P(t+1)$.

2. Listing when action a occurs at location m (measured in discrete time). The state space is $\mathbb{N} \times \mathbb{L}$, where $\mathbb{L}$ denotes the set of lists over $\mathbb{N}$. Look at the process $\lambda_{0,\varnothing}(Q \parallel P(0))$, where $P(t)$ is as in example 1 and Q is given by $Q = \int_{t\geq 0} a(m; t)\cdot Q$.

We have the following definitions:

$effect(tick, \langle n, \sigma\rangle) = \langle n+1, \sigma\rangle$
$action(tick, \langle n, \sigma\rangle) = tick$
$effect(a, \langle n, \sigma\rangle) = \langle n, \sigma * n\rangle$
$action(a, \langle n, \sigma\rangle) = a.$

Moreover we have $m < \ell$. As a consequence, we have e.g.

$effect(a(m)\ \&\ tick(\ell), \langle n, \sigma\rangle) = \langle n+1, \sigma*(n+1)\rangle.$

3. Serial switch. We can draw the following picture of a serial switch (fig. 2).

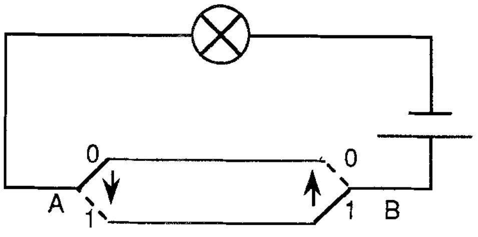

FIGURE 2.

The switches A and B are given by the equations

$$A = \int_{t\geq 0} \mathrm{switch}(\mathrm{left};\, t)\cdot A$$

$$B = \int_{t\geq 0} \mathrm{switch}(\mathrm{right};\, t)\cdot B$$

(here, we have locations left, right corresponding to the positions of the switches). We have state space $S = \{\mathrm{on}, \mathrm{off}\}$. E.g. in figure 2, the lamp is in state off. We define the action and effect functions as follows.

$$\mathrm{action}(\mathrm{switch}, \mathrm{off}) = \mathrm{turnon}$$
$$\mathrm{action}(\mathrm{switch}, \mathrm{on}) = \mathrm{turnoff}$$
$$\mathrm{effect}(\mathrm{switch}, \mathrm{on}) = \mathrm{off}$$
$$\mathrm{effect}(\mathrm{switch}, \mathrm{off}) = \mathrm{on}$$

Starting in state off (as in the figure) we have the process

$$P = \lambda_{\mathrm{off}}(A \,\|\, B).$$

(We assume no communication occurs.)

An interesting situation occurs if both switches are turned at the same time. Suppose we have the ordering right < left. Let t_0 be a fixed time. Then we have

$$\lambda_{\mathrm{off}}(\mathrm{switch}(\mathrm{left}; t_0)\cdot A \,\|\, \mathrm{switch}(\mathrm{right}; t_0)\cdot B) = (\mathrm{turnon}(\mathrm{left}) \,\&\, \mathrm{turnoff}(\mathrm{right}))(t_0)\cdot\lambda_{\mathrm{off}}(A \,\|\, B),$$
$$\lambda_{\mathrm{on}}(\mathrm{switch}(\mathrm{left}; t_0)\cdot A \,\|\, \mathrm{switch}(\mathrm{right}; t_0)\cdot B) = (\mathrm{turnoff}(\mathrm{left}) \,\&\, \mathrm{turnon}(\mathrm{right}))(t_0)\cdot\lambda_{\mathrm{on}}(A \,\|\, B).$$

4.4 EXAMPLE

As a larger example, we present a version of the Concurrent Alternating Bit Protocol. We base our description on the specification in VAN GLABBEEK & VAANDRAGER [GV89]. Other descriptions can be found in KOYMANS & MULDER [KM90], LARSEN & MILNER [LM87].

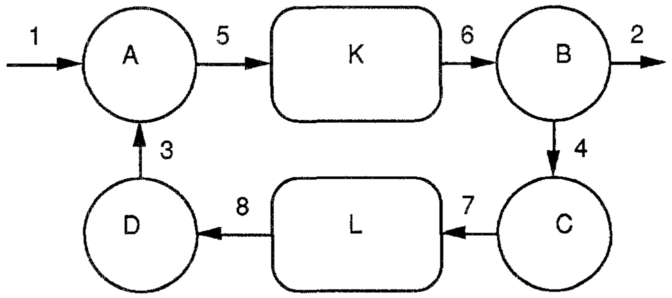

FIGURE 3.

In figure 3, elements of a finite data set D are sent from location 1 to location 2. From A to B, frames consisting of a data element and an alternating bit are sent, from C to D, independently, acknowledgements. K and L are unbounded faulty queues, that can lose data. It was shown in [GV89] that we can assume that loss of data only occurs at the top of the queue. We will model the queues by means of a state operator. We present the specification first, and then define the state operator.

In the specification, there are two parameters: w_0 is the amount of time that a sender allows to pass before a message is sent into the queue again, and w_1 is the amount of time after which a receiver checks the queue again after an unsuccessful attempt. Put another way, $\frac{1}{w_0}$ is the retransmission frequency, and $\frac{1}{w_1}$ is the polling frequency. These parameters can be filled in arbitrarily, and do not affect the correctness of the protocol. Moreover, in the specification, we use a system delay constant 1.

The specification of the senders and receivers now looks as follows:

$A = A(0)$

$A(b) = \int_{t\geq 0} \sum_{d\in D} r(d)(1; t)\cdot enq(db)(5; t+1)\cdot A(d, b, t+2)$ for each $b \in \{0, 1\}$

$A(d, b, t) = enq(db)(5; t+w_0)\cdot A(d, b, t+w_0) + \int_{v\in(t,t+w_0]} r(next)(3; v)\cdot A(1-b)$

for each $b\in\{0,1\}, d\in D, t\geq 0$.

$B = B(0, 0)$

$B(b, t) = \big(deq(\perp)(6; t+w_1) + \sum_{d\in D} deq(d(1-b))(6; t+w_1)\big)\cdot B(b, t+w_1) +$

$+ \sum_{d\in D} deq(db)(6; t+w_1)\cdot B(d, b, t+w_1)$ for each $b \in \{0,1\}, t \geq 0$.

$B(d, b, t) = s(d)(2; t+1)\cdot s(next)(4; t+2)\cdot B(1-b, t+2)$ for each $b\in\{0,1\}, d\in D, t\geq 0$.

$C = C(1, 0)$

$C(b, t) = enq(b)(7; t+w_0)\cdot C(b, t+w_0) + \int_{v\in(t,t+w_0]} r(next)(4; v)\cdot enq(7; v+1)\cdot C(1-b, v+2)$

for each $b \in \{0,1\}, t \geq 0$.

$D = D(0, 0)$

$D(b, t) = \big(deq(\perp)(8, t+w_1) + deq(1-b)(8; t+w_1)\big)\cdot D(b, t+w_1) +$

$+ deq(b)(8; t+w_1)\cdot s(next)(3; t+w_1+1)\cdot D(1-b, t+w_1+1)$ for each $b \in \{0, 1\}, t \geq 0$.

$E = \int_{t\geq 0} error(6; t)\cdot E$ $\qquad$ $F = \int_{t\geq 0} error(8; t)\cdot F$

Now the Concurrent Alternating Bit Protocol is defined by:

$CABP = \partial_H(\lambda_\varnothing(A \| E \| B) \| \lambda_\varnothing(C \| F \| D))$.

Here, the encapsulation set is

$H = \{r(next)(k), s(next)(k) : k\in \{2, 3\}\}$.

In the process CABP, we are actually dealing with two state operators: the state operator for processes A, E, B has a state space consisting of lists of frames, the state operator for C, F, D has a state space consisting of lists of booleans. We have the following ordering on locations: $6 > 5$ and $8 > 7$. This makes it impossible to add an element to an empty queue and read it out at the same instant of time: an element needs a positive amount of time to propagate through the queue.

In order to save space, we define the action and effect function for both state operators at the same time (so in the following, either $x \in D\times B$ or $x \in B$, and either $\sigma \in (D\times B)^*$ or $\sigma \in B^*$):

$action(enq(x), \sigma) = c(x)$

$effect(enq(x), \sigma) = \sigma^* x$

$action(deq(\perp), \sigma) = i$ if $\sigma = \varnothing$

$action(deq(\perp), \sigma) = \delta$ if $\sigma \neq \varnothing$

$action(deq(x), \sigma) = \delta$ if $\sigma = \varnothing$ or $top(\sigma) \neq x$

$action(deq(x), \sigma) = c(x)$ if $top(\sigma) = x$

$effect(deq(x), \sigma) = \sigma$ if $\sigma = \varnothing$ or $top(\sigma) \neq x$

effect(deq(x), σ) = tail(σ)	if top(σ) = x
action(error, $\varnothing$) = δ	
action(error, σ) = i	if $\sigma \neq \varnothing$
effect(error, $\varnothing$) = $\varnothing$	
effect(error, σ) = tail(σ)	if $\sigma \neq \varnothing$.

Now we believe that this specification constitutes a correct protocol, if the queues behave fairly (i.e. data do not get lost infinitely many times in a row). At this point, we will not state explicitly what this statement means (leaving this for further research), so we will not give a specification that process CABP satisfies after abstraction of internal actions (the internal actions are the c(x) actions and i).

Note that multi-actions can occur in this protocol also: we can add to a non-empty queue and read from it at the same time.

5. CONCLUSION

We conclude that it is possible to add a state operator to real time process algebra, and that we can specify interesting examples by means of it. We claim that the state operator will be useful to describe communication between a sender and a moving receiver.

REFERENCES

[BB88] J.C.M. BAETEN & J.A. BERGSTRA, *Global renaming operators in concrete process algebra,* Information & Computation 78 (3), 1988, pp. 205-245.

[BB91a] J.C.M. BAETEN & J.A. BERGSTRA, *Real time process algebra,* Formal Aspects of Computing 3 (2), 1991, pp. 142-188. (Report version appeared as report P8916, University of Amsterdam 1989.)

[BB91b] J.C.M. BAETEN & J.A. BERGSTRA, *Real space process algebra,* in Proc. CONCUR'91, Amsterdam (J.C.M. Baeten & J.F. Groote, eds.), Springer LNCS 527, 1991, pp. 96-110. To appear in Formal Aspects of Computing.

[BW90] J.C.M. BAETEN & W.P. WEIJLAND, *Process algebra,* Cambridge Tract in TCS 18, Cambridge University Press 1990.

[BK84] J.A. BERGSTRA & J.W. KLOP, *Process algebra for synchronous communication,* Inf. & Control 60, 1984, pp. 109-137.

[BK86] J.A. BERGSTRA & J.W. KLOP, *Process algebra: specification and verification in bisimulation semantics,* in: Math. & Comp. Sci. II (M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens, eds.), CWI Monograph 4, North-Holland, Amsterdam 1986, pp. 61-94.

[GV89] R.J. VAN GLABBEEK & F.W. VAANDRAGER, *Modular specifications in process algebra (with curious queues,* in: Algebraic Methods: Theory, Tools & Applications, Passau 1987 (M. Wirsing & J.A. Bergstra, eds.), Springer LNCS 394, 1989, pp. 465-506.

[HE88] M. HENNESSY, *Algebraic theory of processes,* MIT Press 1988.

[H85] C.A.R. HOARE, *Sequential communicating processes,* Prentice Hall, 1985.

[J91a] A. JEFFREY, *A linear time process algebra,* in Proc. 3rd CAV, (K.G. Larsen & A. Skou, eds.), Aalborg 1991, report IR-91-4/5, Aalborg University, pp. 501-512.

[J91b] A. JEFFREY, *Abstract timed observation and process algebra,* in Proc. CONCUR'91, Amsterdam (J.C.M. Baeten & J.F. Groote, eds.), Springer LNCS, 1991, pp. 332-345.

[K91] A.S. KLUSENER, *Completeness in real time process algebra,* in Proc. CONCUR'91, Amsterdam (J.C.M. Baeten & J.F. Groote, eds.), Springer LNCS, 1991, pp. 376-392.

[KM90] C.P.J. KOYMANS & J.C. MULDER, *A modular approach to protocol verification using process algebra,* in: Applications of Process Algebra (J.C.M. Baeten, ed.), Cambridge Tracts in TCS 17, Cambridge University Press 1990, pp. 261-306.

[LM87] K.G. LARSEN & R. MILNER, *A complete protocol verification using relativized bisilulation,* in: Proc. 14th ICALP, Karlsruhe (Th. Ottmann, ed.), Springer LNCS 267, 1987, pp. 126-135.

[M80] R. MILNER, *A calculus of communicating systems,* Springer LNCS 92, 1980.

[M89] R. MILNER, *Communication and concurrency,* Prentice Hall, 1989.

[MT90] F. MOLLER & C. TOFTS, *A temporal calculus of communicating systems,* in: Proc. CONCUR'90, Amsterdam (J.C.M. Baeten & J.W. Klop, eds.), Springer LNCS 458, 1990, pp. 401-415.

[MU91] D.V.J. MURPHY, *Testing, betting and timed true concurrency,* in Proc. CONCUR'91, Amsterdam (J.C.M. Baeten & J.F. Groote, eds.), Springer LNCS, 1991, pp. 439-454.

[RR88] G.M. REED & A.W. ROSCOE, *A timed model for communicating sequential processes,* TCS 58, 1988, pp. 249-261.

[S90] SPECS CONSORTIUM, *Definition of MR and CRL version 2.1,* 1990.

[V90] F.W. VAANDRAGER, *Process algebra semantics of POOL,* in: Applications of Process Algebra (J.C.M. Baeten, ed.), Cambridge Tracts in TCS 17, Cambridge University Press 1990, pp. 173-236.